word distance exchange 2

```
vector<int> minDistance(string &word1, string &word2) {
    vector<int> ans;

    int size1 = word1.size();
    int size2 = word2.size();

    if (size1 != size2)
        return ans;

    unordered_map<char, int> map1;
    unordered_map<char, unordered_map<char, int>> map2;
    int i;

    for (i = 0; i < size1; i++) {
        if (word1[i] == word2[i])
            continue;

        if (map2.count(word1[i])) {
            if (map2[word1[i]].count(word2[i])) {
                return vector<int>({map2[word1[i]][word2[i]], i});
            } else {
                ans = {map2[word1[i]].begin()->second, i};
            }
        } else {
            if (map1.count(word2[i])) {
                ans = {map1[word2[i]], i};
            }
        }

        map1[word1[i]] = i;
        map2[word2[i]][word1[i]] = i;
    }

    return ans;
}
```

bike

```
    def assignBikes(self, workers, bikes):
        def dis(i, j):
            return abs(workers[i][0] - bikes[j][0]) + abs(workers[i][1] -
bikes[j][1])
        h = [[0, 0, 0]]
```

```
        seen = set()
        while True:
            cost, i, taken = heapq.heappop(h)
            if (i, taken) in seen: continue
            seen.add((i, taken))
            if i == len(workers):
                return cost
            for j in xrange(len(bikes)):
                if taken & (1 << j) == 0:
                    heapq.heappush(h, [cost + dis(i, j), i + 1, taken | (1 << j)])
```

Find in Mountain Array

```
class Solution {
    int findInMountainArray(int target, MountainArray A) {
        int n = A.length(), l, r, m, peak = 0;
        // find index of peak
        l  = 0;
        r = n - 1;
        while (l < r) {
            m = (l + r) / 2;
            if (A.get(m) < A.get(m + 1))
                l = peak = m + 1;
            else
                r = m;
        }
        // find target in the left of peak
        l = 0;
        r = peak;
        while (l <= r) {
            m = (l + r) / 2;
            if (A.get(m) < target)
                l = m + 1;
            else if (A.get(m) > target)
                r = m - 1;
            else
                return m;
        }
        // find target in the right of peak
        l = peak;
        r = n - 1;
        while (l <= r) {
            m = (l + r) / 2;
            if (A.get(m) > target)
                l = m + 1;
            else if (A.get(m) < target)
```

```
            r = m - 1;
        else
            return m;
    }
    return -1;
  }

}
```

KMP

```
class Solution {
public:
    int strStr(string haystack, string needle) {
        int m = haystack.size(), n = needle.size();
        if (!n) {
            return 0;
        }
        vector<int> lps = kmpProcess(needle);
        for (int i = 0, j = 0; i < m;) {
            if (haystack[i] == needle[j]) {
                i++, j++;
            }
            if (j == n) {
                return i - j;
            }
            if (i < m && haystack[i] != needle[j]) {
                j ? j = lps[j - 1] : i++;
            }
        }
        return -1;
    }
private:
    vector<int> kmpProcess(string needle) {
        int n = needle.size();
        vector<int> lps(n, 0);
        for (int i = 1, len = 0; i < n;) {
            if (needle[i] == needle[len]) {
                lps[i++] = ++len;
            } else if (len) {
                len = lps[len - 1];
            } else {
                lps[i++] = 0;
            }
        }
        return lps;
    }
};
```

Rectangel Sum MIN

1292. Maximum Side Length of a Square with Sum Less than or Equal to Threshold

```
public int maxSumSubmatrix(int[][] matrix, int k) {
    if (matrix == null || matrix.length == 0 || matrix[0].length == 0)
        return 0;
    int rows = matrix.length, cols = matrix[0].length;
    int[][] areas = new int[rows][cols];
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            int area = matrix[r][c];
            if (r-1 >= 0)
                area += areas[r-1][c];
            if (c-1 >= 0)
                area += areas[r][c-1];
            if (r-1 >= 0 && c-1 >= 0)
                area -= areas[r-1][c-1];
            areas[r][c] = area;
        }
    }
    int max = Integer.MIN_VALUE;
    for (int r1 = 0; r1 < rows; r1++) {
        for (int r2 = r1; r2 < rows; r2++) {
            TreeSet<Integer> tree = new TreeSet<>();
            tree.add(0);    // padding
            for (int c = 0; c < cols; c++) {
                int area = areas[r2][c];
                if (r1-1 >= 0)
                    area -= areas[r1-1][c];
                Integer ceiling = tree.ceiling(area - k);
                if (ceiling != null)
                    max = Math.max(max, area - ceiling);
                tree.add(area);
            }
        }
    }
    return max;
}
```

587. Erect the Fence

straing num 5

```
class Solution(object):
    def isNStraightHand(self, hand, W):
        if len(hand) % W != 0: return False
        count = collections.Counter(hand)
        while count:
            m = min(count.keys())
            num = count[m]
```

```
            for k in range(m, m+W):
                v = count[k]
                if v < num: return False
                if v == num:
                    del count[k]
                else:
                    count[k] = v - num

        return True
```

Campus bike

```
class Solution {
    public int[] assignBikes(int[][] workers, int[][] bikes) {
        int n = workers.length;
        Queue<int[]> q = new PriorityQueue<int[]>((a, b)->(a[0] == b[0] ? (a[1] == b[1] ? a[2] - b[2] : a[1] - b[1]) : a[0] - b[0]));
        int i = 0;
        for (int[] worker : workers) {
            int j = 0;
            for (int[] bike : bikes) {
                q.add(new int[]{Math.abs(bike[0] - worker[0]) + Math.abs(bike[1] - worker[1]), i, j++});
            }
            i++;
        }
        int[] res = new int[n];
        Arrays.fill(res, -1);
        Set<Integer> visited = new HashSet<>();
        while (visited.size() < n) {
            int[] temp = q.poll();
            if (res[temp[1]] == -1 && !visited.contains(temp[2])) {
                res[temp[1]] = temp[2];
                visited.add(temp[2]);
            }
        }
        return res;
    }
}
```

Campus Bike2

```
    def assignBikes(self, workers, bikes):
        def dis(i, j):
            return abs(workers[i][0] - bikes[j][0]) + abs(workers[i][1] - bikes[j][1])
```

```
        h = [[0, 0, 0]]
        seen = set()
        while True:
            cost, i, taken = heapq.heappop(h)
            if (i, taken) in seen: continue
            seen.add((i, taken))
            if i == len(workers):
                return cost
            for j in xrange(len(bikes)):
                if taken & (1 << j) == 0:
                    heapq.heappush(h, [cost + dis(i, j), i + 1, taken | (1 <<
j)])
```

947. Most Stones Removed with Same Row or Column

```
class Solution {
  Map<Integer, Integer> f = new HashMap<>();
  int islands = 0;

  public int removeStones(int[][] stones) {
    for (int i = 0; i < stones.length; ++i)
      union(stones[i][0], ~stones[i][1]);
    return stones.length - islands;
  }

  public int find(int x) {
    if (f.putIfAbsent(x, x) == null)
      islands++;
    if (x != f.get(x))
      f.put(x, find(f.get(x)));
    return f.get(x);
  }

  public void union(int x, int y) {
    x = find(x);
    y = find(y);
    if (x != y) {
      f.put(x, y);
      islands--;
    }
  }
}
```

91. Decode Ways

1096. Brace Expansion II

803. Bricks Falling When Hit

727. Minimum Window Subsequence

count-complete-tree-nodes

1145. Binary Tree Coloring Game

685. Redundant Connection II